Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks

B. Chandrasekaran

Laboratory for Artificial Intelligence Research
The Ohio State University
Columbus, OH 43210 USA

Abstract

The level of abstraction of much of the work in knowledge-based systems (the rule, frame, logic level) is too low to provide a rich enough vocabulary for knowledge and control. I provide an overview of a framework called the Generic Task approach that proposes that knowledge systems should be built out of building blocks, each of which is appropriate for a basic type of problem solving. Each generic task uses forms of knowledge and control strategies that are characteristic to it, and are generally conceptually closer to domain knowledge. This facilitates knowledge acquisition, and can produce a more perspicuous explanation of problem solving. The relationship of the constructs at the generic task level to the rule-frame level is analogous to that between high level programming languages and assembly languages. 1 describe a set of generic tasks that have been found particularly useful in constructing diagnostic, design and planning systems; diagnostic reasoning is used to illustrate the approach. 1 describe the Generic Task Toolset for constructing knowledge systems, which embodies the Generic Task approach. I conclude with the implications of this approach for the functional architecture of intelligence.

1. Overview of the Paper

The first part of the paper is a critique of the level of abstraction in much of the current discussion on knowledge-based systems. It will be argued that the level of rules-logic-frames-networks is inappropriate for discussing many issues of knowledge organization and control. We advocate instead the level of abstraction associated with the language of generic tasks, types of knowledge, and types of control strategies.

Following this I will outline the elements of a framework for the design of knowledge-based systems that we have been developing in our laboratory over the last several years. Complex knowledge-based reasoning tasks can often be decomposed into a number of *generic tasks, each associated with certain types of knowledge and a family of control strategies.* At each stage in the reasoning, the system will engage in one of the generic tasks, depending upon the knowledge available and the state of problem solving. Diagnostic reasoning will be used to illustrate the application of these ideas. I will discuss the advantages of this approach for knowledge acquisition, knowledge representation, control of problem solving, and explanation. These advantages are made possible by the richer vocabulary in terms of which knowledge and control are represented for each task.

I then describe how the above approach leads naturally to a new technology: a toolset which helps one to build expert systems by using higher level building blocks. 1 will review the toolset, and discuss the advantages that accrue from its use. Finally, I discuss what this approach entails for the architecture of intelligence, and discuss a number of related theoretical issues.

These ideas have evolved over the years as a result of work in diagnostic and design problem solving. A number of earlier publications (Chandrasekaran 1983, 1984, 1986) trace the development of the ideas.

2. Critique of Uniform Architectures

Knowledge representation has been a major concern of AI, in particular and quite naturally, for knowledge-based problem solving (or "expert systems[11]). The general assumption, and consequently the methodology, has been that there is something called domain knowledge that needs to be acquired—quite independent of the problems one might wish to solve—and that the role of the knowledge representation formalism is to help encode it. Logics of various kinds, rule-based languages and frame representations have been three popular kinds of proposals for knowledge representation. Each of these representations has a natural family of inference mechanisms that can operate on it. Each of the knowledge representations, along with an inference scheme that is appropriate for it, defines an *architecture.* When the inference scheme is fixed, the representation formalism is also said to provide a *shell* for inserting knowledge.

These architectures (with relatively small additions, if needed) are computationally universal. Thus the important point about building knowledge systems

with them is not whether a task can be performed, but whether they offer knowledge and control constructs that are natural to the task. All of these (and other similar) languages fall short when one considers tasks of some complexity such as planning or diagnosis.

2.1. Lack of Expressiveness for Higher Level Tasks

The level of abstraction of these languages obscures the essential nature of the information processing that is needed for the performance of higher level tasks. They are like knowledge system assembly languages, rather than programming languages with constructs for capturing the essence of the information processing phenomena.

Intuitively one thinks that there are types of knowledge and control strategies that are common to diagnostic reasoning in different domains, and similarly that there are common structures and strategies for, say, design as a cognitive activity; but that the structures and control strategies for diagnostic reasoning and design problem solving will generally be different. However, when one looks at the formalisms (or equivalently the languages) that are commonly used in expert system design, the knowledge representation and control strategies do not typically capture these distinctions. For example in diagnostic reasoning one might generically wish to speak in terms of malfunction hierarchies, rule-out strategies, setting up a differential, etc., while for design the generic terms might be device/component hierarchies, design plans, ordering of subtasks, etc. Ideally one would like to represent diagnostic knowledge in a domain by using the vocabulary that is appropriate for the task, but the languages in which the expert systems have been implemented have sought uniformity across tasks, and thus have had to lose perspicuity of representation at the task level.

In addition, the control strategies that these languages come with (such as forward or backward chaining for rule systems) do not explicitly indicate the real control structure of a task that a problem solver is performing. For example, the fact that RI (McDermott 1982) performs a linear sequence of subtasks as a way of performing its design task is not explicitly encoded. This task-specific control structure is "encrypted" and hence invisible at the level of the pattern-matching control of OPS5. The knowledge base of a system that is built in one of these architectures ends up accumulating a large number of programming devices as if they were part of domain knowledge. This detracts from the modularity of domain knowledge, since debugging a piece of knowledge involves studying the interaction between domain knowledge and the task as mediated by the implicit programming knowledge in the knowledge base.

2.2. Artifactual versus Real Issues

Because of the mismatch between architecture and information processing task, control issues arise that are artifacts of the architecture, but are often misinterpreted as issues having to do with control at the task level. For example rule-based approaches often concern themselves with conflict resolution strategies. Yet if the knowledge were viewed at a different level, one can often abstract *organizations* of knowledge that only a small, highly relevant body of knowledge is brought up, without any need for conflict resolution at all. In many rule-based systems, these organizational constructs can be "programmed" in the rule language by the use of context setting rules and metarules, but because the rules and metarules, per se, have been considered to be knowledge-level phenomena (as opposed to the implementation-level phenomena, which they often are), knowledge acquisition has often been directed towards strategies for conflict resolution, when they ought to be directed to issues of knowledge organization.

In sum, these architectures, by encouraging knowledge acquisition and representation at a level far removed from the organization and control of a task, create barriers in building understandable, modifiable knowledge systems.

3. Generic Tasks

Our work is based an alternative view, *viz.,* that *knowledge representation and use cannot be separated* (Gomez and Chandrasekaran 1981). That is, knowledge should be in different forms, depending upon the type of function for which it is used. By "use" I do not mean a highly domain-specific thing such as, "This piece of knowledge will be useful in the treatment of cancer," but a more generic problem solving use that can be applied to a variety of domains. The following information specifies the generic task abstractly.

- The *function* of the task. What type of problem does it solve? What is the nature of the information that it takes as input, and produces as output?
- The *representation and organization of knowledge.* What are the primitive terms in which the forms of knowledge needed for the task can be represented? How should knowledge be organized and structured for that task?
- The *control strategy.* What control strategy (inference strategy) can be applied to the knowledge to accomplish the function of the generic task?

3.1. Generic Tasks and Generic Task Problem Solvers

In order to understand how problem solvers are built using generic tasks, we can think of a task specification in accordance with the above as a virtual specification of a *shell.* When domain knowledge is

encoded using the primitive terms, organization and structure that are specified for the task, and then combined with the inference strategy that comes with the task, we have a *generic task problem solver.* The totality of domain knowledge is *distributed* among a number of such problem solving modules, representing a variety of generic tasks. Thus a Generic Task problem solver is not merely an information processing strategy: it is a inference strategy that uses knowledge to solve parts of the problem. The interaction between modules is based on their information needs and functions: a module which needs information of a certain type gets it from a module whose function matches it.

4. Generic Tasks in Diagnostic Reasoning

The motivation in our work on diagnosis is to make a connection between diagnostic problem solving, and the general properties of intelligence as information processing strategies. I will now describe a functional architecture approach to performing diagnosis, which uses and integrates information provided by a number of generic problem solvers. This approach originated with the MDX system (Chandrasekaran and Mittal 1983), and has been refined in the construction of the RED system (Smith *et al.* 1985, Josephson *et al.* 1987].

Let us view diagnostic problem solving as a task in which the goal is to explain a set of observations of a system. The explanation is to be in terms of malfunctions that may have caused the observations. Since the malfunctions can interact in a number of ways, the problem solver has to produce a set of malfunction hypotheses that explain all the data without including unnecessary hypotheses and also taking into account the possible interactions between hypotheses.

Let us consider domains satisfying the following properties:

1. Knowledge is available in the form of malfunction hierarchies mirroring the class/subclass relationship among the malfunctions.
2. Given a typical observation, only a relatively small subset of these malfunctions could be implicated. Also interactions between malfunctions with respect to an observation are limited.

Property 2 requires that sufficient variety of observations are available from several parts of the system to substantially disambiguate the diagnostic situation. This property holds true in medicine and in a number of mechanical domains. De Kleer's work on diagnosing multiple malfunctions (de Kleer and Williams 1986) involves a situation where the worst-case assumptions about interactions apply. In those domains, such as digital circuits, Property 2 does not hold.

The knowledge available in domains satisfying these properties makes possible a decomposition of problem solving into two submodules: a *hierarchical classifier* which uses the hierarchy to select a small set of plausible malfunctions, and an *abductive assembler* that uses a subset of these hypotheses to make a composite that provides a coherent and parsimonious explanation of data. The malfunction hierarchy has more general classes of malfunctions in its higher level nodes and more specialized ones as their successors, e.g, "Liver" has as a successor, "hepatitis." The forms of knowledge for the former module use terms in which hierarchies are described, while the forms of knowledge for the latter module deal with interactions among the hypotheses. I will describe the problem solving behavior of these and other modules in a rather oversimplified manner. References are given where details can be found.

Gomez (Gomez and Chandrsekaran 1981) proposed hierarchical classification as a core process in medical diagnosis, and investigated the inference methods useful for that task. The inference mechanism for hierarchical classification can be thought of as variations on the strategy of *establish-refine,* i.e., the hypothesis space is explored top down, by attempting to establish the top level hypotheses first. When a hypothesis is ruled out, all its more specialized successors can also be ruled out, while if it is established, the conclusion can be refined by considering its successors. This process can be repeated until a number of terminal nodes of the hierarchy are established. This process can be expected to produce a small number of highly plausible hypotheses, each of which explains a portion of the data.

Josephson *et* a/., (Josephson *et al.* 1987) have investigated the inference method needed for abductive assembly. It can be thought of as a means-ends process, driven by the need to explain the most significant unexplained observation at any stage of problem solving. As hypotheses are accumulated to explain the significant observations, the composites are critiqued for redundancy, logical compatibility and so on. Driven by these criteria, a coherent collection of hypotheses that best explain the data is made.

Returning to the classification process, domain knowledge is required to establish or reject each of the hypotheses in the hierarchy. What is needed is a way to decide how well a hypothesis fits a set of observations. This function can be accomplished in a number of ways in different domains depending upon what kind of domain knowledge is available.

A generic strategy called *hypothesis matching* (Chandrasekaran *et al.* 1982, Chandrasekaran 1983) or *structured matching* (Bylander and Johnson 1987) is useful in a number of domains for performing this function. We can build up a hierarchy of abstractions

from the data to the hypothesis. For example, given the hypothesis, "Liver disease," one way to decide on the degree of fit between the data and the hypothesis is first to see how the data match the intermediate abstractions, "Chemical test evidence," "physical evidence," and "historical evidence." Each of the abstractions will normally involve a subset of the data applicable to Liver as a whole. Each step in the abstractions can be computed by pattern matching, and the values passed up for further matching. This pattern matching is actually implemented in our work as tables analogous to Samuel's Signature Tables (Samuel 1967), but the details are not important for the current discussion. For each of the hypotheses in the hierarchy, a structured matcher can be built using the scheme.

The matcher requires values for specific data items. In the above example, data relevant to a decision about "historical evidence" might be "evidence of exposure to anesthetics?", or "fever responds to drugs?". In the first example, the associated database might have the information that the patient had undergone major surgery a few days earlier, and in the latter, the data base may have the complete information about the patients prescriptions and temperatures. In either case, the needed information can be obtained by making appropriate inferences from domain knowledge: about the relationship between anesthetics and surgery in the first example, and some complex processing of raw data to make the abstraction about fever response in the latter. Thus there is need for an inferencing database to make the necessary data abstractions. I will not go into the details of the representation and inference control here, save to note that they are different from the representation and inference for the other problem solvers: hierarchical classification, abductive assembly and structured matching. Mittal (Mittal *tt al.* 1984) recognized the inferencing database as an important component generic activity in the design of MDX.

The overall problem solving for diagnosis can now be traced as follows. The hierarchical classifier's problem solving activity proceeds top down, and for each hypothesis that is considered, the structured matcher for that hypothesis is invoked for information about the degree of fit with the data. The structured matcher turns to the inferencing database for information about the data items that it is interested in. The database completes its reasoning and passes on the needed information to the structured matcher. After acquiring all the needed data from the database in a similar fashion, the structured matcher completes its matching activity for that hypothesis and returns that value of the match to the classifier, along with the data that the hypothesis can explain. The classifier's activity now proceeds along the lines of its control strategy: i.e., it either rules out the hypothesis, or establishes it and pursues the successors. This process of each problem solver invoking other problem solvers who can provide the information needed for the performance of its own task is repeated until the classifier concludes with a number of high plausibility hypotheses, and information about what each of them can explain. At this point, the abductive assembler takes over and proceeds to construct the composite explanatory hypothesis for the problem.

These are not the only problem solvers that could be useful for diagnosis, of course. Additional problem solvers with their own types of knowledge and control can be helpful. If the classificatory structure or the structured matcher is incomplete in its knowledge, case-based reasoners (Kolodner and Simpson 1986), or deeper domain models such as qualitative reasoners and functional reasoners (Sembugamoorthy and Chandrasekaran 1986) can be invoked. MDX2 (Sticklen 1987) is an example of a diagnostic system whose classifier interacts with a cognitive deep model of parts of its domain for obtaining information about the relationship between observations and hypotheses.

The four generic tasks that we have just described (hierarchical classification, abductive assembly, structured matching, and database inference) are all distinct: the knowledge representation and inference method for each is distinctly different. They are also generic in the sense that each could be used by any other problem solver needing its functionality in that domain.

The description that I just gave of problem solving omits many of the subtleties in the inference strategies of each of the problem solvers, since my aim was to introduce the methodology rather than present the complete theory of diagnosis. However, the following additional points are relevant since they refer to the important role played by the generic task architecture.

1. The interaction between the abductive assembler and the classifier may be much more dynamic, where the refinement of medium confidence hypotheses in the classificatory structure can be done at the command of the assembler, which is looking for hypotheses to explain some unexplained data.

2. The control of classificatory behavior in the presence of additive and subtractive observations can be complicated. For these and other complexities in control of classification problem solving, see (Sticklen *tt al.* 1985).

3. Multiple and tangled hierarchies can be incorporated as is being done in MDX2 (Sticklen 1987). Since a particular hypothesis can occur in more than one hierarchy, that is, it can be classified according to more than one perspective, the restriction of the hierarchies to tree structures is not burdensome or unnatural.

4. Because of the property that for each observation, only a small number of malfunctions at each level of the hierarchy can be implicated, multiple malfunctions are a natural for this architecture. Interactions are allowed as long as they can be reasoned about during classification (Gomez and Chandrasekaran 1981) and assembly within the general assumption of near-independence.

5. Test ordering and data validation: In the description of the approach I have assumed that all the data are available. However, the architecture makes it easy to focus the test ordering decisions in a natural way. For each hypothesis knowledge is available about what tests are useful for it to be established or rejected, and as mentioned in 2 above the assembler can propose that hypotheses which were not strongly enough established for refinement be refined further in the interests of explaining remaining observations. This may actually call for additional tests to establish or reject the successors. Also, the assembler will often be able to identify equally plausible, but alternative, hypotheses. In order to resolve them, additional tests may be necessary. This architecture makes possible test ordering that can be driven by the goals of each of the tasks.

   The localization of goals and knowledge that is helpful in test ordering can also be employed to provide a form of know ledge-based sensor validation, since sensors that conflict in their contribution to a hypothesis can be located and critiqued. See (Chandrasekaran and Punch 1987) for further details.

4.1. The Fallacy of Surface Phenominalism

If one were to look at the behavior of the diagnostic system described above, without any awareness of its internal architecture of generic tasks, almost all of the standard architectures could be ascribed to it, depending upon the level of abstraction at which the behavior is observed. As the hypothesis matcher is working through the hierarchical abstraction, it would appear to be a rule processor using evidences on the antecedent side to reach a conclusion about the intermediate abstractions or the hypothesis. At this level it would also appear to be a data-directed activity. At the classifier level, the system may seem to be a frame system moving from hypothesis concept to hypothesis concept. At this level it would appear to be a hypothesis-directed activity. The term "the fallacy of surface phenominalism" refers to the pitfalls that are possible in going from external problem solving behavior, such as protocols of human experts, directly to an architecture.

5. Conceptualization and Design of the Generic Task Toolset

Each of the generic tasks can be used as a programming technique within a more general programming language like LISP, PROLOG, or OPS5. However, this does not prevent an knowledge engineer from going outside the boundaries of a generic task. These bounds are important to specify because they ensure that the advantages of generic tasks will be maintained. One natural way to do this is to implement a software tool for each generic task to be used in a general programming environment. Such tools also provide an empirical means for testing the clarity of these ideas and the usefulness of the approach in actual systems.

We have been motivated by the problems of diagnosis, design and planning in developing our toolset. In addition to the generic tasks that were described in connection with the diagnosis, we have found two other generic tasks very useful for our purposes: *object synthesis using plan selection and refinement* (Brown and Chandrasekaran 1986) for certain classes of design problems, and *state abstraction* (Chandrasekaran 1983) for certain types of prediction of system-level consequences as a result of state changes to subsystems. Due to space limitations, I do not describe them here.

For each generic task, we have developed a tool that can encode the problem solving and knowledge that is appropriate for the task. Below is a list of the tools that correspond to the generic tasks that we have studied: *CSRL* (Conceptual Structures Representation Language) is the tool for hierarchical classification (Bylander and Mittal 1986); *DSPL* (Design Specialists and Plans Language) is the tool for object synthesis using plan selection and refinement (Brown and Chandrasekaran 1986); *ID ABLE* (Intelligent DAta Base LanguagE) is the tool for knowledge-directed information passing (Sticklen 1983); *HYPER* (HYPothesis matchER) is the tool for hypothesis matching (Johnson and Josephson 1986); *PEIRCE* (named after the philosopher C. S. Peirce) is the tool for abductive assembly of hypotheses (Punch *et al.* 1986); and *WW HI* (What Will Happen If) is the tool for state abstraction.

The tools are intended to ensure the following advantages of the generic tasks:

- *Multiformity.* The more traditional architectures for the construction of knowledge based systems emphasize the advantages of uniformity of representation and inference. However, in spite of the advantage of simplicity, we argued earlier that uniformity results in a level of abstraction problem. A uniform representation cannot capture important distinctions between different kinds of knowledge-use needs. A uniform inference engine does not provide different control structures for different kinds of problems.

The generic task approach provides multiformity. Each generic task provides a different way to organize and use knowledge. The knowledge engineer can choose which generic task is the best for performing a particular function, or can use different generic tasks for performing the same function. Different problems can use different generic tasks and different combinations of generic tasks.

- *Modularity.* A knowledge-based system can be designed by making a functional decomposition of its intended problem solving into several cooperating generic tasks, as illustrated in our discussion on diagnosis. Each generic task provides a way to decompose a particular function into its conceptual parts, e.g., the categories for hierarchical classification, and allows domain knowledge of other forms to be inserted into a generic task, e.g., evidence combination knowledge in hierarchical classification (Sticklen 1987). Each generic task localizes the knowledge that is used to satisfy local goals.

- *Knowledge Acquisition.* Each generic task is associated with its own knowledge acquisition strategy for building an efficient problem solver (Bylander and Chandrasekaran 1987). For example in hierarchical classification, the knowledge engineer needs to find out what specific categories should be contained in the classification hierarchy and what general categories provide the most leverage for the establish-refine strategy.

- *Explanation.* This approach directly helps in providing explanations of problem solving in expert systems in two important ways: how the data match local goals and how the control strategy operates (Chandrasekaran *et al.* 1987). Also, the control strategy of each generic task is specific enough for generating explanations of why the problem solver chose to evaluate or not to evaluate a piece of knowledge. This is because of the higher level of abstraction in which control is specified for generic tasks.

- *Exploiting Interaction between Knowledge and Inference.* Rather than trying to separate knowledge from its use, each generic task specifically integrates a particular way of representing knowledge with a particular way of using knowledge. This allows the attention of the knowledge engineer to be focused on representing and organizing knowledge for performing problem solving.

- *Tractability.* Under reasonable assumptions, each generic task generally provides tractable problem solving (Allemang *et al.* 1987, Goel *et al.* 1987). (One major exception is abductive assembly, which can become intractable under certain conditions, making it hard then for humans and machines to perform the task.) The main reasons why they are tractable are that a problem can be decomposed into small, efficient units, and knowledge can be organized to take care of combinatorial interactions in advance.

It should be noted that these advantages are attained at the cost of generality. Each generic task is purposely constrained to perform a limited type of problem solving and requires the availability of appropriate domain knowledge.

The Generic Task toolset is implemented as a collection of specialists which interact by passing messages for control and information exchange. This is a natural implementation at the level of generic tasks. As is turns out, the problem solving activities *within* a generic task problem solver are also implemented in our toolset as a collection of specialists; e.g., within the hierarchical classifier, each of the classificatory hypotheses is a problem solving module in its own right.

How the different generic problem solvers interact is an active research issue. The current theory and the toolset are based on each of the problem solvers explicitly invoking another problem solver for needed information. I believe that a more attractive long-term approach would be one where the problem solver broadcasts the need for some information, and other problem solvers which can deliver the information respond to the request. This reduces the degree of explicitness needed at system design time. This also increases the possibility that a piece of knowledge or a module added somewhere in the system will be able to contribute its problem solving power without the designer needing to foresee this possibility.

5.1. Abstract Representation of Control:

It is often stated that using rule-based approaches makes it possible to have control knowledge explicitly represented as collections of rules. This declarative form of representation of control is said to help in reasoning about, modifying or explaining the control behavior of the system.

The GT problem solvers are active agents. Thus it might appear that the advantages of explicitness of control may be lost in the approach. As it turns out, the control strategies associated with each of the generic task architectures are implemented as a family of *message types.* The user can modify them within limits permissible for each generic task and the explanation generation system uses the content of the messages directly in its description of the control behavior. The fact that the messages have both a procedural content to them as well as a declarative representation gives them all the advantages of abstract representation of control.

6. Generic Tasks and Other Use-Specific Architectures

In the late 70's, when we embarked on this line of research — characterised by an attempt to identify generic tasks and the forms knowledge and control required to perform them — the dominant paradigms in knowledge-based systems were rule and frame type architectures. While our work on use-specific architectures was evolving, dissatisfaction at the limited vocabulary of tasks that these architectures were offering was growing at other research centers. Clancey in particular noted the need for specifying the information processing involved by using a vocabulary of higher level tasks. Task-level architectures have been gathering momentum lately: McDermott and his coworkers (Marcus and McDermott 1987) have built SALT, a shell for a class of design problems, where critiquing proposed designs by checking for constraint-violations is applicable. Clancey (Clancey 1985) has proposed a shell called Heracles which incorporates a strategy for diagnosis: he calls it *heuristic classification.* Bennett (Bennett 1986) presents COAST, a shell for the design of configuration problem solving systems. Gruber and Cohen (Gruber and Cohen 1987) offer a system called MUM for *managing uncertainty* and so on. All these approaches share the basic thesis of our own work, viz., the need for task-specific analyses and architecture support for the task. However, there are some differences in assumptions and methodology in some cases that needs further discussion. Further, once we identify task-level architectures as the issue for highest leverage, then the immediate question is: what is the criterion by which a task is deemed to be not only generic but is appropriate for modularization as an architecture? How about an architecture for the generic task of "investment decisions"? Diagnosis? Diagnosis of process control systems? Is uncertainty management a task for which it will be useful to have an architecture? Are we going to proliferate a chaos of architectures without any real hope of reuse? What are the possible relationship between these architectures? Which of these architectures can be built out of other architectures? I do not propose to answer all these questions here, but they seem to be the appropriate kinds of questions to ask when one moves away from the comfort of universal architectures and begins to work with different architectures for different problems.

At this stage in the development of these ideas, empirical investigation of different proposals from the viewpoint of usefulness, tractability and composability is the best strategy. From a practical viewpoint, any architecture that has a useful function and for which one can identify knowledge primitives and an inference method ought to be considered a valid candidate for experimentation. As the tools evolve, one may find that some of the architectures are further decomposable into equally useful, but more primitive, architectures; or that some of them do not represent particularly useful functionalities, and so on. Nevertheless, the following distinctions can be made on conceptual grounds, and can be used to drive the empirical investigations.

- "Building blocks" out of which more complex problem solvers can be composed, such as the tasks in the theory presented earlier in the paper.
- Explicit high level strategies which we want a system to follow, where the strategies are expressed in terms of some set of tasks. Heuristic Classification is an example.
- Compound tasks, such as the form of diagnosis described in earlier in the paper. An architecture for this compound task will bring with it its constituent generic tasks and also show how to integrate them from the viewpoint of the overall task.
- Tasks which do not necessarily correspond to those human experts do well, but nevertheless can be captured as appropriate combinations of knowledge and inference and a clear function can be associated with them, e.g., constraint satisfaction schemes.

I want to examine how the tasks in these different senses relate to the generic task theory.

Heuristic classification (Clancey 1985) is a strategy in the sense of its being an appropriate behavior for diagnosis, i.e., it is a collection of tasks to be performed to accomplish the goal. Heraeles, the architecture that supports heuristic classification, uses metarules as a way of programming this strategy in a rule system: the metarules represent abstractly the control behavior that would be required for each of the tasks in the strategy. Heracles as a shell will enable the designer to build diagnostic systems using all or portions of the above strategy. In our work, we wish to get as much of the strategy as possible to *emerge* from the interaction of more elementary problem solvers. It is not clear that each of the tasks in a higher level strategy such as heuristic classification necessarily corresponds to one of the problem solvers in our sense. For example, as I have mentioned, the *data abstraction* part of his strategy actually emerges in our architecture for diagnosis from the abstraction step in the hypothesis matcher and data-to-data reasoning processes in the knowledge-directed data base. From a theoretical viewpoint this is not an irrelevant distinction: Our theoretical goal is an "atomic" theory of knowledge use, so that more complex problem solving behaviors can be seen to emerge from the interaction of such atoms. From a practical viewpoint, while our architecture produces many of the needed behaviors in an emergent manner, the constituent parts still may need to be integrated to make the overall system

produce the needed behavior. Thus, direct study of such "compound" problem solving behaviors such as heuristic classification is illuminating and technically useful. Eventually, however, if the atomic theory is right, it will be able to show how the subtasks in heuristic classification arise from the form in which domain knowledge is available, and also provide a more principled vocabulary of subtasks in behaviors: e.g., what's "group and differentiate" and where does it come from?

Regarding compound tasks, 1 need to point out that some of the generic tasks in our repertoire, such as *object synthesis by plan selection and refinement,* seem to me to be more complex than others and represent several generic tasks organized and integrated for a certain purpose. The DSPL planner, e.g., has plan selectors which use structured matching. The current set in our laboratory has evolved empirically. As we experimentally discover decomposability, we will proceed to separate compound tasks into simpler constituent tasks. We are also building even higher level architectures for practically important applications. For example, we are currently building a diagnostic shell which will integrate the problem solvers as outlined in our discussion. The shell can then be directly used for the design and implementation of diagnostic systems of this particular type.

There may be significant technological value in generic tasks whose function is to solve problems which human experts find it difficult to do without pencil and paper or computers, if they are integrated appropriately with other generic tasks. Constraint satisfaction problems are of this type. Design by constraint satisfaction is a useful method, but humans are not in general very good at solving problems in this way. The only caution in using this form of reasoning for design is that such architectures may encourage formulating design problems which have interesting and useful decompositions other than as constraint satisfaction problems, causing difficulties in debugging and explanation, i.e., they may preclude a study of domain knowledge that helps a human designer in producing explainable und maintainable, if only satisficing, designs.

7. What Makes a Building Block

Clearly, highly domain specific tasks, such as a shell for designing drug-therapy administration systems, are not generic in an interesting AI sense, though they may be generic within the domain (e.g., the drug-therapy shell can be instantiated for different drugs).

The work of Cohen and his coworkers (Gruber and Cohen 1987) in regard to the MUM system (and its successor MU for uncertainty management) raises interesting issues related to this. It incorporates a strategy for explicitly reasoning about balancing uncertainty reduction with costs of tests to reduce uncertainty. They regard diagnostic reasoning as an instance of the uncertainty management problem. On the other hand, it can also be argued that uncertainty management is a component of diagnostic problem solving. Clearly all problem solving, whether design or diagnosis, can be thought of as instances of uncertainty reduction.

It seems to me that there are clearly situations in which managing uncertainty requires explicit and conscious strategies, and MU can be useful in those cases. However, it seems unlikely that uncertainty management is a generic building block activity in the sense of this paper, since the forms and strategies for handling uncertainty seem to be conditioned by the demands of the problem being solved and the form in which knowledge is available. The structured matcher handles uncertainty in one fashion, while the classifier deals with it in another. In (Chandrasekaran and Tanner 1986), 1 have discussed this view that uncertainty handling is not a unitary activity. I can see MU serving as a local advisor for uncertainty decisions within each of the generic tasks, when it is felt necessary to make some decisions, say test ordering, requiring explicit uncertainty manipulation.

8. Towards a Functional Architecture of Intelligence

The generic tasks that are represented in our toolset were specifically chosen to be useful as technology for building diagnosis, planning and design systems with compiled expertise. For capturing intelligent problem solving in general, we will undoubtedly require many more such elementary strategies and ways of integrating thern. For example, the problem solving activities in qualitative reasoning and device understanding, e.g., qualitative simulation, consolidation, and functional representation, qualify as generic problem solving activities, as do weak methods such as *means-ends analysis.* The work of Schank and his associates has also generated a body of such representations and inference processes that have a generic character to them. All these tasks have well-defined information processing functions, specific knowledge representation primitives and inference methods. Thus candidates for generic information processing modules in our sense are indeed many. What does all this mean for an architecture of intelligence?

I am led to a view of intelligence as an interacting collection of *functional units,* each of which solves an information processing problem by using knowledge in a certain form and corresponding inference methods that are appropriate. Each of these units defines an information processing faculty. I discuss elsewhere (Chandrasekaran 1987) the view that these functional units share a computational property: they provide the agent with the means of transforming essentially

intractable problems into versions which can be solved efficiently by using knowledge and inference in certain forms. For example, Goel *et* a/., (Goel *et al.* 1987) show how classification problem solving solves applicable cases of diagnosis with low complexity, while diagnosis in general is of high complexity. Knowledge is indeed power, but how it acquires its power is a far subtler story than the first generation knowledge based systems made it appear.

This view generates its own research agenda: As a theory, the generic tasks idea has quite a bit of work ahead of it to tell a coherent story about how the tasks come together and are integrated, and how more complex tasks such as planning come about from more elementary ones. How complex inference methods develop from simpler ones and how learning shapes these functional modules, are issues to be investigated.

The theory does not take a position on the information processing architecture over which these functional units may be defined, i.e., this is not an argument for or against the architecture of the substratum being realized as a rule processor or as a frame system. In fact, a continuing issue in AI is how the rule/frame viewpoints may be integrated within a principled framework.

The generic tasks idea has strong implications to knowledge representation and suggests a view of what mentalese, the language of thought, might be. What ultimately characterizes the generic task approach is not the proposals for specific generic tasks, which will undoubtedly evolve empirically, but a commitment to the following view: Knowledge and its use need to be specified together in knowledge representation. Because *how* knowledge is used depends upon the *form* in which knowledge appears, the enterprise of knowledge representation is one of developing vocabularies simultaneously for the expression and use of knowledge. The languages in which each of the generic information processing units encode their knowledge and communicate their information needs and solutions collectively defines the language of thought.

Acknowledgments

I acknowledge the central role grants 82-0255 and 87-0090 from Air Force Office of Scientific Research have played in the development of these ideas. The support from Defense Advanced Research Projects Agency, RADC Contract F 30602-85-C-0010, has also been invaluable. Battelle University Grants Program helped in the development of portions of the tool box. Fd like to acknowledge the contributions of Fernando Gomez, Sanjay Mittal, Jack Smith, David Brown, John Josephson, Tom Bylander, Jon Sticklen, and my current graduate students to the evolution of these ideas. The help of Ashok Goel and Cindy Sergent in the preparation of this paper is. gratefully acknowledged.

References

1. Allemang, D,, M. C. Tanner, T. Bylander, and J. R. Josephson. "On the Computational Complexity of Hypothesis Assembly". To appear in Proceedings 1JCAI-87, Milan, Italy, August 1987.

2. Bennett, J. "COAST: A task-specific tool for reasoning about configurations". In Proceedings AAA] Workshop on High-Level tools, Shawnee Park, Ohio, 1986.

3. Brown, D. C. and B. Chandrasekaran. "Knowledge and Control for Mechanical Design Expert System". IEEE Computer 19(7):92-100, 1986.

4. Bylander, T. and B. Chandrasekaran. "Generic Tasks for Know ledge-Based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition". To Appear in International Journal of Man-Machine Studies, 1987.

5. Bylander, T. and T. R. Johnson. "Structured Matching". Technical Report, Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210. February 1987.

6. Bylander, T. and S. Mittal. "CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling". AI Magazine 7(2):66-77, 1986.

7. Chandrasekaran, B. "Towards a Taxonomy of Problem Solving Types". A] Magazine 4(I):9-17, 1983.

8. Chandrasekaran, B. "Expert Systems: Matching Techniques to Tasks". In Artificial Intelligence Applications for Business, W. Reitman editor, pages 116-132, Ablex, Norwood, New Jersey, 1984.

9. Chandrasekaran, B. "Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design". IEEJE Expert I(3):23-30, 1986.

10. Chandrasekaran, B. "What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and a Proposal". Presented at the AAAI Workshop on Foundations of AI, Las Cruces, New Mexico, February 1986. To appear in Foundations of Artificial Intelligence, D. Partridge editor, 1987.

11. Chandrasekaran, B. and S. Mittal. "Conceptual Representation of Medical Knowledge for Diagnosis: MDX and Related Systems". Advances in Computers, pages 217-293, Academic Press, New York, 1983.

12. Chandrasekaran, B., S. Mittal, and J. W. Smith. "Reasoning with Uncertainty: The MDX Approach". In Proceedings Congress of American Medical Informatics Association, pages 335-339, San Francisco, 1982.

13. Chandrasekaran, B. and W. F. Punch. "Data Validation During Diagnosis: A Step Beyond Traditional Sensor Validation". To appear in Proceedings AAAI-87, Seattle, Washington. July 1987.

14. Chandrasekaran, B. and M. C. Tanner. "Uncertainty Handling in Expert Systems: Uniform Versus Task-Specific Formalisms". In Uncertainty in Artificial Intelligence, L. N. Kanal and J. Lemmer editors, pages 35-46, North Holland, Amsterdam, 1986.

15. Chandrsekaran, B, M. C. Tanner, and J. R. Josephson. "Explanation: The Role of Control Strategies and Deep Models", In Expert Systems: The User Interface, Ablex, Norwood, New Jersey, 1987.

16. Clancey, W. J. "Heuristic Classification". Artificial Intelligence 27(3):289-350, 1985.

17. de Kleer, J. and B. C. Williams. "Reasoning about Multiple Faults". In Proceedings AAAI-86, pages 132-139, Philadelphia, Pennsylvania, August 1986.

18. Goel, A., N. Soundararajan and B. Chandrasekaran. "Complexity in Classificatory Reasoning". To appear in Proceedings Seattle, Washington, July 1987.

19. Gomez, F. and B. Chandrasekaran. "Knowledge Organization and Distribution for Medical Diagnosis". IEEE Transactions on Systems, Man and Cybernetics SMC-II(I):34-42, 1981.

20. Gruber, T. and P. Cohen. "Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition". To appear in International Journal of Man-Machine Studies, 1987.

21. Johnson, T. R. and J. R. Josephson. "HYPER: The Hypothesis Matcher Tool". Technical Report, Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210, 1986.

22. Josephson, J. R., B. Chandrasekaran, J. W. Smith and M. C. Tanner. "A Mechanism for Forming Composite Explanatory Hypotheses". To appear in IEEE Transactions on Systems, Man and Cybernetics. 1987.

23. Kolodner, J. L. and R. L. Simpson. "Problem Solving and Dynamic Memory". In Experience, Memory, and Reasoning, J. L. Kolodner and C. K. Riesberk editors, pages 99-114, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.

24. Marcus, S. and J. McDermott. "SALT: A Knowledge Acquisition Tool for Purpose-and-Revise Systems'[1]. Technical Report, Department of Computer Science, Carnegie-Mellon University. Pittsburg, Pennsylvania, 1987.

25. McDermott, J. "RI: A Rule-Based Configurer of Computer Systems[1]'. Artificial Intelligence 19(I):39-88, 1982.

26. Mittal, S., B. Chandrsekaran, and J. Sticklen. "PATREC: A Knowledge-Directed Database for a Diagnostic Expert System". IEEE Computer 17(9):51-58, 1984.

27. Punch, W. F., M. C. Tanner, and J. R. Josephson. "Design Considerations for PEIRCE: A High Level Language for Hypothesis Assembly". In Proceedings Expert Systems in Government, pages 279-281, McLean, Virginia, 1986.

28. Samuel, A. "Some Studies in Machine Learning Using the Game of Checkers. Recent Progress II". IBM Journal of Research and Development 11(6):601-617, 1967.

29. Sembugamoorthy, V. and B. Chandrasekaran. "Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems". In Experience s Memory,, and Reasoning J- L Kolodner and C. K. Riesbeck editors, pages 47-73, Lawrence Erlbaum, Hillsdale, New Jersey, 1986.

30. Smith, J. W., J. R. Svirbely, C. A. Evans, P. Strohm, J. R. Josephson, and M. C. Tanner. "RED: A Red-Cell Antibody Identification Expert Module". Journal of Medical Systems 9(3):121-138. 1985.

31. Sticklen, J. "Manual for IDABLE". Technical Report, Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210, 1983.

32. Sticklen. J. "MDX2: An Integrated Medical Diagnostic System". Phd. Dissertation in preparation, Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210, 1987.

33. Sticklen, J., B. Chandrasekaran, and J. R. Josephson. "Control Issues in Classificatory Diagnosis". In Proceedings IJCAI-85, pages 300-306, Los Angeles, 1985.